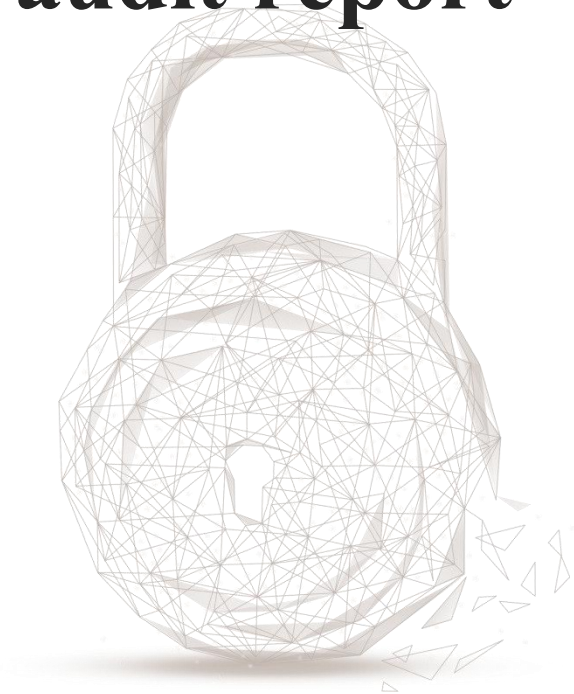
BEOSIN

Blockchain Security

Smart contract security audit report



Audit Number: 202106071836

Report Query Name: BounceNFT

Audit File Info:

Audit File Name	Audit File Hash (SHA256)
BounceNFT.sol	9DD8937F3A4A75952CE17BF80B8F8B2B50CE8671FED897F88C027606053DBC88 (First edition) F0F9B7306638AF87E715B301E825D7EAA88509CCE7EA43447785E9C55EF4EA58(Final edition)
BounceFixedEndEnglishAuctionNFT.sol	B57CAB9D7D2DCD6EC9E700895982329681276CC29C2D6459C8E25587D0B169AE (First edition) 6CE554464C48665DD9879FCD69EB25F642EFCA9DFA837C726FAEA4121E90F8E4(Final edition)
BounceDutchAuction.sol	286F4593D26434247537AFB6A1B577B15CD91B8469B2520138E667B7731C4EEF (First edition) FCE59D546A56269B43041EA2559AE5C50A034FC85E0AFB2F7528F9794691E947(Final edition)
BounceFixedSwap.sol	21EBCFE6C3860CB5C74F8FA763103C806927E91AA10D9F287EDFC965CF49839F (First edition) B720647A2ABDF855BD0B2C1692505EC249029354CF05187BAE4BF1BFE1C7BC19(Final edition)
BounceLottery.sol	265023CB3AB6164FCCFD8532E46AFE4C6E667175C553054995FD23DC613F941A (First edition) 30DD4178AB22A8F7A16014C82E794851B423C11AD8998A0E38A0D5F8DAD001AD(Final edition)
BounceSealedBid.sol	068B62D6BB1167E2931E4A284362F8E43CDF898A02D04B0724C8A15A487C440C (First edition) A538E801FD147AAE10D62B405E41DF8AEFC9E1DA880CDA86095BD2117D59E379(Final edition)

Start Date: 2021.05.21

Completion Date: 2021.06.07

Overall Result: Pass

Audit Team: Beosin (Chengdu LianAn) Technology Co. Ltd.

Audit Categories and Results:

No.	Categories	Subitems	Results
1	Coding Conventions	Compiler Version Security	Pass
		Deprecated Items	Pass
		Redundant Code	Pass
		SafeMath Features	Pass
		require/assert Usage	Pass
		Gas Consumption	Pass
		Visibility Specifiers	Pass
		Fallback Usage	Pass
2	General Vulnerability	Integer Overflow/Underflow	Pass
		Reentrancy	Pass
		Pseudo-random Number Generator (PRNG)	Pass
		Transaction-Ordering Dependence	Pass
		DoS (Denial of Service)	Pass
		Access Control of Owner	Pass
		Low-level Function (call/delegatecall) Security	Pass
		Returned Value Security	Pass
		tx.origin Usage	Pass
		Replay Attack	Pass
		Overriding Variables	Pass
3	Business Security	Business Logics	Pass
		Business Implementations	Pass

of smart contracts, thus taking no responsibility for them. The security audit analysis and other contents of this report are based solely on the documents and materials that the contract provider has provided to Beosin (Chengdu LianAn) Technology before the issuance of this report, and the contract provider warrants that there are no missing, tampered, deleted. if the documents and materials provided by the contract provider are missing, tampered, deleted, concealed or reflected in a situation that is inconsistent with the actual situation, or if the documents and materials provided are changed after the issuance of this report, Beosin (Chengdu LianAn) Technology assumes no responsibility for the resulting loss or adverse effects. The audit report issued by Beosin (Chengdu LianAn) Technology is based on the documents and materials provided by the contract provider, and relies on the technology currently possessed by Beosin (Chengdu LianAn). Due to the technical limitations of any organization, this report conducted by Beosin (Chengdu LianAn) still has the possibility that the entire risk cannot be completely detected. Beosin (Chengdu LianAn) disclaims any liability for the resulting losses.

The final interpretation of this statement belongs to Beosin (Chengdu LianAn).

Audit Results Explained:

Beosin (Chengdu LianAn) Technology has used several methods including Formal Verification, Static Analysis, Typical Case Testing and Manual Review to audit three major aspects of smart contracts project BounceNFT, including Coding Standards, Security, and Business Logic. **The BounceNFT project passed all audit items. The overall result is Pass. The smart contract is able to function properly.**

Audit Contents:

1. Coding Conventions

Check the code style that does not conform to Solidity code style.

1.1 Compiler Version Security

- Description: Check whether the code implementation of current contract contains the exposed solidity compiler bug.
- Result: Pass

1.2 Deprecated Items

- Description: Check whether the current contract has the deprecated items.
- Result: Pass

1.3 Redundant Code

- Description: Check whether the contract code has redundant codes.
- Safety Recommendation: There are many redundant codes in the bounceNFT project, it is recommended to delete them.
- Fixed Result: Fixed.
- Result: Pass

1.4 SafeMath Features

- Description: Check whether the SafeMath has been used. Or prevents the integer overflow/underflow in mathematical operation.
- Result: Pass

1.5 require/assert Usage

- Description: Check the use reasonability of 'require' and 'assert' in the contract.
- Result: Pass

1.6 Gas Consumption

- Description: Check whether the gas consumption exceeds the block gas limitation.
- Result: Pass

1.7 Visibility Specifiers

- Description: Check whether the visibility conforms to design requirement.
- Result: Pass

1.8 Fallback Usage

- Description: Check whether the Fallback function has been used correctly in the current contract.
- Result: Pass

2. General Vulnerability

Check whether the general vulnerabilities exist in the contract.

2.1 Integer Overflow/Underflow

- Description: Check whether there is an integer overflow/underflow in the contract and the calculation result is abnormal.
- Result: Pass

2.2 Reentrancy

- Description: An issue when code can call back into userr contract and change state, such as withdrawing BNB.
- Result: Pass

2.3 Pseudo-random Number Generator (PRNG)

- Description: Whether the results of random numbers can be predicted.
- Result: Pass

2.4 Transaction-Ordering Dependence

- Description: Whether the final state of the contract depends on the order of the transactions.
- Result: Pass

2.5 DoS (Denial of Service)

- Description: Whether exist DoS attack in the contract which is vulnerable because of unexpected reason.
- Result: Pass

2.6 Access Control of Owner

- Description: Whether the owner has excessive permissions, such as malicious issue, modifying the balance of others.
- Result: Pass

2.7 Low-level Function (call/delegatecall) Security

- Description: Check whether the usage of low-level functions like call/delegatecall have vulnerabilities.
- Result: Pass

2.8 Returned Value Security

- Description: Check whether the function checks the return value and responds to it accordingly.
- Result: Pass

2.9 tx.origin Usage

- Description: Check the use secure risk of 'tx.origin' in the contract.
- Result: Pass

2.10 Replay Attack

- Description: Check whether the implement possibility of Replay Attack exists in the contract.
- Result: Pass

2.11 Overriding Variables

- Description: Check whether the variables have been overridden and lead to wrong code execution.
- Result: Pass

3. **Business Security**

3.1 Business analysis of contract BounceNFT

(1) Contract initialization

- Description: As shown in the figure below, the contract implements the *initialize*, *initialize_rinkeby*, *initialize_bsc* and *initialize_heco* functions to initialize the contract (initialize only once). After the contract is deployed, any user can call these functions to set the value of the variable *BotToken*, the variable *MinValueOfBotHolder*, the variable *TxFeeRatio*, and the variable *governor*.

```
818  function initialize(address _governor) public override initializer {
819      require(msg.sender == governor || governor == address(0), "invalid governor");
820      governor = _governor;
821
822      config[TxFeeRatio] = 0.01 ether;
823      config[MinValueOfBotHolder] = 1 ether;
824
825      config[BotToken] = uint(0xA9B1Eb5908CfC3cdf91F9B8B3a74108598009096); // AUCTION
826  }
827
828  function initialize_rinkeby(address _governor) public {
829      initialize(_governor);
830
831      config[BotToken] = uint(0x5E26FA0FE067d28aae8aFf2fB85Ac2E693BD9EfA); // AUCTION
832  }
833
834  function initialize_bsc(address _governor) public {
835      initialize(_governor);
836
837      config[BotToken] = uint(0x1188d953aFC697C031851169EEf640F23ac8529C); // AUCTION
838  }
839
840  function initialize_heco(address _governor) public {
841      initialize(_governor);
842
843      config[BotToken] = uint(address(0)); // AUCTION
844  }
```

Figure 1 Source Code of *initialize*, *initialize_rinkeby, initialize_bsc* and *initialize_heco* Functions

- Related Functions: *initialize, initialize_rinkeby, initialize_bsc, initialize_heco*
- Result: Pass

(2) Create a swap pool

- Description: As shown in the figure below, the contract implements the *createErc721* and *createErc1155* functions for any user to create two different NFT swap pools. The *createErc721* function creates an ERC721 token swap pool by calling the internal function *_create*. the *createErc1155* function creates an ERC1155 token swap pool by calling the internal function *_create*. When creating a swap pool, creator can choose whether to limit purchases to users who hold a certain amount of Bot tokens.

```
846     function createErc721(
847         // name of the pool
848         string memory name,
849         // address of token0
850         address token0,
851         // address of token1
852         address token1,
853         // token id of token0
854         uint tokenId,
855         // total amount of token1
856         uint amountTotal1,
857         // only bot holder can bid the pool
858         bool onlyBot
859     ) external {
860         require(!getDisableErc721(), "ERC721 pool is disabled");
861         if (checkToken0) {
862             require(token0List[token0], "invalid token0");
863         }
864         uint amountTotal0 = 1;
865         _create(
866             name, token0, token1, tokenId, amountTotal0, amountTotal1, onlyBot, TypeErc721
867         );
868     }
```

Figure 2 Source Code of *createErc721* Function

```
870     function createErc1155(
871         // name of the pool
872         string memory name,
873         // address of token0
874         address token0,
875         // address of token1
876         address token1,
877         // token id of token0
878         uint tokenId,
879         // total amount of token0
880         uint amountTotal0,
881         // total amount of token1
882         uint amountTotal1,
883         // only bot holder can bid the pool
884         bool onlyBot
885     ) external {
886         require(!getDisableErc1155(), "ERC1155 pool is disabled");
887         if (checkToken0) {
888             require(token0List[token0], "invalid token0");
889         }
890         _create(
891             name, token0, token1, tokenId, amountTotal0, amountTotal1, onlyBot, TypeErc1155
892         );
893     }
```

Figure 3 Source Code of *createErc1155* Function

- Related Functions: *createErc721, getDisableErc721, createErc1155, getDisableErc1155*
- Result: Pass

(3) Token swap

- Description: As shown in the figure below, the contract implements the *swap* function for users to swap NFT tokens in the existing swap pool. If the *token1* address in the swap pool selected by the user is the zero address (BNB is paid), a 1% handling fee will be charged for this transaction, and the collected handling fee will remain in this contract. If the *token1* address in the swap pool selected by the user is not a zero address, no handling fee will be charged.

```
940  function swap(uint index, uint amount0) external payable
941      isPoolExist(index)
942      checkBotHolder(index)
943      isPoolNotSwap(index)
944  {
945      require(tx.origin == msg.sender, "disallow contract caller");
946      require(!creatorCanceledP[index], "creator has canceled this pool");
947
948      Pool storage pool = pools[index];
949      require(pool.creator != msg.sender, "creator can't swap the pool created by self");
950      require(amount0 >= 1 && amount0 <= pool.amountTotal0, "invalid amount0");
951      require(swappedAmount0P[index].add(amount0) <= pool.amountTotal0, "pool filled or invalid amount0");
952
953      uint amount1 = amount0.mul(pool.amountTotal1).div(pool.amountTotal0);
954      swappedAmount0P[index] = swappedAmount0P[index].add(amount0);
955      swappedAmount1P[index] = swappedAmount1P[index].add(amount1);
956      if (swappedAmount0P[index] == pool.amountTotal0) {
957          // mark pool is swapped
958          swappedP[index] = true;
959      }
960
961      // transfer amount of token1 to creator
962      if (pool.token1 == address(0)) {
963          require(amount1 == msg.value, "invalid ETH amount");
964
965          uint txFee = amount1.mul(getTxFeeRatio()).div(1 ether);
966          uint _actualAmount1 = amount1.sub(txFee);
967          if (_actualAmount1 > 0) {
968              // transfer ETH to creator
969              pool.creator.transfer(_actualAmount1);
970          }
971          if (txFee > 0) {
972              // keep transaction fee in this contract
973              totalTxFee = totalTxFee.add(txFee);
974          }
975      } else {
976          // transfer token1 to creator
977          IERC20(pool.token1).safeTransferFrom(msg.sender, pool.creator, amount1);
978      }
979
980      // transfer tokenId of token0 to sender
981      if (pool.nftType == TypeErc721) {
982          IERC721(pool.token0).safeTransferFrom(address(this), msg.sender, pool.tokenId);
983      } else {
984          IERC1155(pool.token0).safeTransferFrom(address(this), msg.sender, pool.tokenId, amount0, "");
985      }
986
987      emit Swapped(msg.sender, index, amount0, amount1);
988  }
```

Figure 4 Source Code of *swap* Function

- Related Functions: *swap, isPoolExist, checkBotHolder, isPoolNotSwap, getTxFeeRatio, transfer, safeTransferFrom*
- Safety Recommendation: When the *token1* in the swap pool is a zero address, the handling fee *txFee* will always remain in the contract. It is recommended to modify the relevant code, set the handling fee *txFee* processing, or set the function to withdraw this part of BNB.
- Fixed Result: Fixed. Added *withdrawFee* function to withdraw transaction fees.

```
1013  function withdrawFee(address payable to, uint amount) external governance {
1014      totalTxFee = totalTxFee.sub(amount);
1015      to.transfer(amount);
1016  }
```

Figure 5 Source Code of *withdrawFee* Function

- Result: Pass

(4) Cancel swap pool

- Description: As shown in the figure below, the contract implements the *cancel* function for the creator of the swap pool to cancel the specified trading pool. After the swap pool is cancelled, NFT tokens will be directly returned to the creator.

```
990   function cancel(uint index) external
991       isPoolExist(index)
992       isPoolNotSwap(index)
993   {
994       require(isCreator(msg.sender, index), "sender is not pool creator");
995       require(!creatorCanceledP[index], "creator has canceled this pool");
996       creatorCanceledP[index] = true;
997
998       Pool memory pool = pools[index];
999       if (pool.nftType == TypeErc721) {
1000          IERC721(pool.token0).safeTransferFrom(address(this), pool.creator, pool.tokenId);
1001      } else {
1002          IERC1155(pool.token0).safeTransferFrom(address(this), pool.creator, pool.tokenId, pool.amountTotal0.sub
              (swappedAmount0P[index]), "");
1003      }
1004
1005      emit Canceled(msg.sender, index, pool.amountTotal0.sub(swappedAmount0P[index]));
1006  }
```

Figure 6 Source Code of *cancel* Function

- Related Functions: *cancel, isPoolExist, isPoolNotSwap, isCreator, safeTransferFrom*
- Result: Pass

(5) Token0 whitelist check

- Description: When creating a pool, if the variable *checkToken0* is true, it will check whether the *token0* is in the *token0List*.

- Safety Recommendation: The boolean variable *checkToken0* is not initialized and cannot be changed. When creating the ERC721 and ERC1155 pools, *token0* will never be checked. It is recommended to add the *checkToken0* modification function.
- Fixed Result: Fixed.

```
1057    function enableCheckToken0(bool enable) external governance {
1058        checkToken0 = enable;
1059    }
1060
1061    function addToken0List(address[] memory token0List_) external governance {
1062        for (uint i = 0; i < token0List_.length; i++) {
1063            token0List[token0List_[i]] = true;
1064        }
1065    }
1066
1067    function removeToken0List(address[] memory token0List_) external governance {
1068        for (uint i = 0; i < token0List_.length; i++) {
1069            delete token0List[token0List_[i]];
1070        }
1071    }
```

Figure 7 Source Code of *enableCheckToken0*, *addToken0List* and *removeToken0List* Functions

- Result: Pass

(6) Receive NFT token transfer

- Description: As shown in the figure below, the contract implements *onERC721Received* and *onERC1155Received* functions to receive ERC721 and ERC1155 token transfers.

```
1049    function onERC721Received(address, address, uint, bytes calldata) external override returns (bytes4) {
1050        return this.onERC721Received.selector;
1051    }
1052
1053    function onERC1155Received(address, address, uint, uint, bytes calldata) external pure returns(bytes4) {
1054        return this.onERC1155Received.selector;
1055    }
```

Figure 8 Source Code of *onERC721Received* and *onERC1155Received* Functions

- Related Functions: *onERC721Received, onERC1155Received*
- Result: Pass

(7) Query functions

- Description: The contract implements functions such as *getTxFeeRatio, getMinValueOfBotHolder, getBotToken, getDisableErc721, getDisableErc1155* functions to query the relevant data in the contract.
- Related Functions: *getTxFeeRatio, getMinValueOfBotHolder, getBotToken, getDisableErc721, getDisableErc1155*

- Result: Pass

3.2 Business analysis of contract BounceFixedEndEnglishAuctionNFT

(1) Contract initialization

- Description: As shown in the figure below, the contract implements the *initialize*, *initialize_rinkeby*, *initialize_bsc* and *initialize_heco* functions to initialize the contract (initialize only once). After the contract is deployed, any user can call these functions to set the value of the variable *BotToken*, the variable *MinValueOfBotHolder*, the variable *TxFeeRatio*, and the variable governor.

```
829  function initialize(address _governor) public override initializer {
830      super.initialize(_governor);
831
832      unlocked = 1;
833      config[TxFeeRatio] = 0.01 ether;
834      config[MinValueOfBotHolder] = 1 ether;
835
836      config[BotToken] = uint(0xA9B1Eb5908CfC3cdf91F9B8B3a74108598009096); // AUCTION
837  }
838
839  function initialize_rinkeby(address _governor) public {
840      initialize(_governor);
841
842      config[BotToken] = uint(0x5E26FA0FE067d28aae8aFf2fB85Ac2E693BD9EfA); // AUCTION
843  }
844
845  function initialize_bsc(address _governor) public {
846      initialize(_governor);
847
848      config[BotToken] = uint(0x1188d953aFC697C031851169EEf640F23ac8529C); // AUCTION
849  }
850
851  function initialize_heco(address _governor) public {
852      initialize(_governor);
853
854      config[BotToken] = uint(address(0)); // AUCTION
855  }
```

Figure 9 Source Code of *initialize, initialize_rinkeby, initialize_bsc* and *initialize_heco* Functions

- Related Functions: *initialize, initialize_rinkeby, initialize_bsc, initialize_heco*
- Result: Pass

(2) Create a swap pool

- Description: As shown in the figure below, the contract implements the *createErc721* and *createErc1155* functions for any user to create two different NFT swap pools. The *createErc721* function creates an ERC721 token swap pool by calling the internal function *_create*. the *createErc1155* function creates an ERC1155 token swap pool by calling the internal function *_create*. When creating a

swap pool, creator can choose whether to limit purchases to users who hold a certain amount of Bot tokens and set the auction start price, maximum price, duration, price increase and other parameters.

```
857  function createErc721(
858      // name of the pool
859      string memory name,
860      // address of token0
861      address token0,
862      // address of token1
863      address token1,
864      // token id of token0
865      uint tokenId,
866      // maximum amount of token1
867      uint amountMax1,
868      // minimum amount of token1
869      uint amountMin1,
870      // minimum incremental amount of token1
871      uint amountMinIncr1,
872      // reserve amount of token1
873      uint amountReserve1,
874      // duration
875      uint duration,
876      // only bot holder can bid the pool
877      bool onlyBot
878  ) public {
879      require(!getDisableErc721(), "ERC721 pool is disabled");
880      uint tokenAmount0 = 1;
881      _create(name, token0, token1, tokenId, tokenAmount0, amountMax1, amountMin1, amountMinIncr1, amountReserve1,
         duration, onlyBot, TypeErc721);
882  }
```

Figure 10 Source Code of *createErc721* Function

```
884     function createErc1155(
885         // name of the pool
886         string memory name,
887         // address of token0
888         address token0,
889         // address of token1
890         address token1,
891         // token id of token0
892         uint tokenId,
893         // amount of token id of token0
894         uint tokenAmount0,
895         // maximum amount of token1
896         uint amountMax1,
897         // minimum amount of token1
898         uint amountMin1,
899         // minimum incremental amount of token1
900         uint amountMinIncr1,
901         // reserve amount of token1
902         uint amountReserve1,
903         // duration
904         uint duration,
905         // only bot holder can bid the pool
906         bool onlyBot
907     ) public {
908         require(!getDisableErc1155(), "ERC1155 pool is disabled");
909         _create(name, token0, token1, tokenId, tokenAmount0, amountMax1, amountMin1, amountMinIncr1,
            amountReserve1, duration, onlyBot, TypeErc1155);
910     }
```

Figure 11 Source Code of *createErc1155* Function

- Related Functions: *createErc721, getDisableErc721, createErc1155, getDisableErc1155*
- Result: Pass

(3) Bid for NFT tokens

- Description: As shown in the figure below, the contract implements the *bid* function for users to bid for NFT tokens. The user can call this function to participate in the auction, and the user's bid must not be lower than the current bid plus a minimum increase. After this bid is successful, the tokens paid by the last bidder will be returned. If the current bid is greater than the maximum auction price, the auction ends and the auction is automatically executed.

```
982     function bid(
983         // pool index
984         uint index,
985         // amount of token1
986         uint amount1
987     ) external payable
988         lock
989         isPoolExist(index)
990         checkBotHolder(index)
991         isPoolNotClosed(index)
992     {
993         address payable sender = msg.sender;
994
995         Pool storage pool = pools[index];
996         require(tx.origin == msg.sender, "disallow contract caller");
997         require(pool.creator != sender, "creator can't bid the pool created by self");
998         require(amount1 != 0, "invalid amount1");
999         require(amount1 >= pool.amountMin1, "the bid amount is lower than minimum bidder amount");
1000        require(amount1 >= currentBidderAmount(index), "the bid amount is lower than the current bidder amount");
1001
1002        if (pool.token1 == address(0)) {
1003            require(amount1 == msg.value, "invalid ETH amount");
1004        } else {
1005            IERC20(pool.token1).safeTransferFrom(sender, address(this), amount1);
1006        }
1007
1008        // return ETH to previous bidder
1009        if (currentBidderP[index] != address(0) && currentBidderAmount1P[index] > 0) {
1010            if (pool.token1 == address(0)) {
1011                currentBidderP[index].transfer(currentBidderAmount1P[index]);
1012            } else {
1013                IERC20(pool.token1).safeTransfer(currentBidderP[index], currentBidderAmount1P[index]);
1014            }
1015        }
1016
1017        // record new winner
1018        currentBidderP[index] = sender;
1019        currentBidderAmount1P[index] = amount1;
1020        bidCountP[index] = bidCountP[index] + 1;
1021        myBidderAmount1P[sender][index] = amount1;
1022
1023        emit Bid(sender, index, amount1);
1024
1025        if (pool.amountMax1 > 0 && pool.amountMax1 <= amount1) {
1026            _creatorClaim(index);
1027            _bidderClaim(sender, index);
1028        }
1029    }
```

Figure 12 Source Code of *bid* Function

- Related Functions: *bid, isPoolExist, checkBotHolder, safeTransfer, isPoolNotSwap, isContract, currentBidderAmount, safeTransferFrom, transfer*
- Safety Recommendation: 1)When the *token1* in the swap pool is a zero address, the handling fee *txFee* will always remain in the contract. It is recommended to modify the relevant code, set the handling fee *txFee* processing, or set the function to withdraw this part of BNB. 2)If the user who created the pool or the user who participated in the auction is a contract, and the type of tokens paid is BNB, then the user contract can refuse to receive BNB to prevent refunds, which will cause the corresponding tokens to be locked in the contract. It is recommended to determine whether the user who created the pool or the user who participated in the auction is a contract. 3)The *bid* function can be called repeatedly consecutively by the same address. It is recommended to confirm whether this situation needs to be avoided.
- Fixed Result: 1)Fixed. Added *withdrawFee* function to withdraw transaction fees. 2)Fixed. The contract is forbidden to call related functions. 3)Ignored.

```
1080  function withdrawFee(address payable to, uint amount) external governance {
1081      totalTxFee = totalTxFee.sub(amount);
1082      to.transfer(amount);
1083  }
```

Figure 13 Source Code of *withdrawFee* Function

- Result: Pass

(4) Claim tokens

- Description: As shown in the figures below, the contract implements *creatorClaim* and *bidderClaim* functions for the creator and user of the trading pool to withdraw tokens. If the auction is successful (the final bid is greater than the lowest auction price), the user receives NFT tokens, and the creator of the trading pool receives the tokens paid by the user(1% handling fee will be charged). If the auction fails, each will receive the tokens paid.

```
1031  function creatorClaim(uint index) external
1032      isPoolExist(index)
1033      isPoolClosed(index)
1034  {
1035      require(isCreator(msg.sender, index), "sender is not pool's creator");
1036      _creatorClaim(index);
1037  }
```

Figure 14 Source Code of *creatorClaim* Function

```
1073  function bidderClaim(uint index) external
1074      isPoolExist(index)
1075      isPoolClosed(index)
1076  {
1077      _bidderClaim(msg.sender, index);
1078  }
```

Figure 15 Source Code of *bidderClaim* Function

- Related Functions: *creatorClaim, bidderClaim, isPoolExist, isPoolNotSwap, isCreator*
- Result: Pass

(5) Receive NFT token transfer

- Description: As shown in the figure below, the contract implements *onERC721Received* and *onERC1155Received* functions to receive ERC721 and ERC1155 token transfers

```
1117    function onERC721Received(address, address, uint256, bytes calldata) external override
        returns (bytes4) {
1118        return this.onERC721Received.selector;
1119    }
1120
1121    function onERC1155Received(address, address, uint256, uint256, bytes calldata) external
        pure returns(bytes4) {
1122        return this.onERC1155Received.selector;
1123    }
```

Figure 16 Source Code of *onERC721Received* and *onERC1155Received* Function

- Related Functions: *onERC721Received, onERC1155Received*
- Result: Pass

(6) Query functions

- Description: The contract implements functions such as *getPoolCount, currentBidderAmount, getTxFeeRatio, getMinValueOfBotHolder, getDisableErc1155 and getBotToken* to query the relevant data in the contract.
- Related Functions: *getTxFeeRatio, getMinValueOfBotHolder, getBotToken, getDisableErc721, getDisableErc1155*
- Result: Pass

3.3 Business analysis of contract BounceDutchAuction

(1) Contract initialization

- Description: As shown in the figure below, the contract implements the *initialize* and *initialize_rinkeby* functions to initialize the contract (initialize only once). After the contract is deployed, any user can call these functions to set the value of the variable *BotToken*, the variable *MinValueOfBotHolder*, the variable *TxFeeRatio*, and the variable *StakeContract*.

```
109    function initialize() public initializer {
110        super.__Ownable_init();
111        super.__ReentrancyGuard_init();
112
113        config[TxFeeRatio] = 0.015 ether;
114        config[MinValueOfBotHolder] = 60 ether;
115        config[BotToken] = uint(0xA9B1Eb5908CfC3cdf91F9B8B3a74108598009096);
116        config[StakeContract] = uint(0x98945BC69A554F8b129b09aC8AfDc2cc2431c48E);
117    }
118
119    function initialize_rinkeby() public {
120        initialize();
121
122        config[BotToken] = uint(0x5E26FA0FE067d28aae8aFf2fB85Ac2E693BD9EfA);
123        config[StakeContract] = uint(0xa77A9FcbA2Ae5599e0054369d1655D186020ECE1);
124    }
```

Figure 17 Source Code of *initialize* and *initialize_rinkeby* Functions

- Related Functions: *initialize, initialize_rinkeby*
- Result: Pass

(2) Create a swap pool

- Description: As shown in the figure below, the contract implements the *create* function for users to create a swap pool. The creator can put any BEP-20 token as a commodity in the swap pool. When creating a swap pool, creator can choose whether to limit purchases to users who hold a certain amount of Bot tokens and set the auction start price, maximum price and other parameters. The creator can also decide who can make a purchase by setting a whitelist.

```
118    function create(CreateReq memory poolReq, address[] memory whitelist_) public payable
119        nonReentrant
120        isPoolNotCreate(poolReq.creator)
121    {
122        require(poolReq.amountTotal0 != 0, "the value of amountTotal0 is zero");
123        require(poolReq.amountMin1 != 0, "the value of amountMax1 is zero");
124        require(poolReq.amountMax1 != 0, "the value of amountMin1 is zero");
125        require(poolReq.amountMax1 > poolReq.amountMin1, "amountMax1 should larger than amountMin1");
126        require(poolReq.duration != 0, "the value of duration is zero");
127        require(poolReq.duration <= 7 days, "the value of duration is exceeded one week");
128        require(poolReq.times != 0, "the value of times is zero");
129        require(bytes(poolReq.name).length <= 15, "the length of name is too long");
130
131        uint index = pools.length;
```

Figure 18 Source Code of *create* Function (1/2)

```
133         // transfer amount of token0 to this contract
134         IERC20  _token0 = IERC20(poolReq.token0);
135         uint token0BalanceBefore = _token0.balanceOf(address(this));
136         _token0.safeTransferFrom(poolReq.creator, address(this), poolReq.amountTotal0);
137         require(
138             _token0.balanceOf(address(this)).sub(token0BalanceBefore) == poolReq.amountTotal0,
139             "not support deflationary token"
140         );
141         // reset allowance to 0
142         _token0.safeApprove(address(this), 0);
143 
144         if (whitelist_.length > 0) {
145             enableWhiteList = true;
146             for (uint i = 0; i < whitelist_.length; i++) {
147                 whitelistP[index][whitelist_[i]] = true;
148             }
149         }
150 
151         // creator pool
152         Pool memory pool;
153         pool.name = poolReq.name;
154         pool.creator = poolReq.creator;
155         pool.token0 = poolReq.token0;
156         pool.token1 = poolReq.token1;
157         pool.amountTotal0 = poolReq.amountTotal0;
158         pool.amountMax1 = poolReq.amountMax1;
159         pool.amountMin1 = poolReq.amountMin1;
160         pool.times = poolReq.times;
161         pool.duration = poolReq.duration;
162         pool.openAt = poolReq.openAt;
163         pool.closeAt = poolReq.openAt.add(poolReq.duration);
164         pools.push(pool);
165 
166         if (poolReq.onlyBot) {
167             onlyBotHolderP[index] = poolReq.onlyBot;
168         }
169 
170         myCreatedP[poolReq.creator] = pools.length;
171 
172         emit Created(index, msg.sender, pool);
173     }
```

Figure 19 Source Code of *create* Function (2/2)

- Related Functions: *create, isPoolNotCreate, balanceOf, safeTransferFrom*
- Safety Recommendation: Whether the whitelist check is enabled or not is determined by the user who created the swap pool. As long as there is a user who entered the whitelist_ when the creation is not empty, the whitelist will be enabled. If some pools are created without entering a whitelist, no one can redeem them. It is recommended to set a separate whitelist flag for each pool.
- Fixed Result: Fixed.

```
126     function create(CreateReq memory poolReq, address[] memory whitelist_) external nonReentrant {
127         require(tx.origin == msg.sender, "disallow contract caller");
128         require(poolReq.amountTotal0 != 0, "the value of amountTotal0 is zero");
129         require(poolReq.amountMin1 != 0, "the value of amountMax1 is zero");
130         require(poolReq.amountMax1 != 0, "the value of amountMin1 is zero");
131         require(poolReq.amountMax1 > poolReq.amountMin1, "amountMax1 should larger than amountMin1");
132         require(poolReq.openAt <= poolReq.closeAt && poolReq.closeAt.sub(poolReq.openAt) < 7 days, "invalid closed")
133         require(poolReq.times != 0, "the value of times is zero");
134         require(bytes(poolReq.name).length <= 15, "the length of name is too long");
135
136         uint index = pools.length;
137
138         // transfer amount of token0 to this contract
139         IERC20  _token0 = IERC20(poolReq.token0);
140         uint token0BalanceBefore = _token0.balanceOf(address(this));
141         _token0.safeTransferFrom(poolReq.creator, address(this), poolReq.amountTotal0);
142         require(
143             _token0.balanceOf(address(this)).sub(token0BalanceBefore) == poolReq.amountTotal0,
144             "not support deflationary token"
145         );
146
147         if (poolReq.enableWhiteList) {
148             require(whitelist_.length > 0, "no whitelist imported");
149             _addWhitelist(index, whitelist_);
150         }
151
152         // creator pool
153         Pool memory pool;
154         pool.name = poolReq.name;
155         pool.creator = poolReq.creator;
156         pool.token0 = poolReq.token0;
157         pool.token1 = poolReq.token1;
158         pool.amountTotal0 = poolReq.amountTotal0;
159         pool.amountMax1 = poolReq.amountMax1;
160         pool.amountMin1 = poolReq.amountMin1;
161 //          pool.amountReserve1 = poolReq.amountReserve1;
162         pool.times = poolReq.times;
163         pool.duration = poolReq.closeAt.sub(poolReq.openAt);
164         pool.openAt = poolReq.openAt;
165         pool.closeAt = poolReq.closeAt;
166         pool.enableWhiteList = poolReq.enableWhiteList;
167         pools.push(pool);
168
169         if (poolReq.onlyBot) {
170             onlyBotHolderP[index] = poolReq.onlyBot;
171         }
172
173         myCreatedP[poolReq.creator] = pools.length;
174
175         emit Created(index, msg.sender, pool);
176     }
```

Figure 20 Source Code of *create* Function (Fixed)

- Result: Pass

(3) Bid for BEP-20 tokens

- Description: As shown in the figure below, the contract implements the *bid* function for users to bid for designated BEP-20 tokens. The user can call this function to participate in the auction, and the user's bid must not be lower than the price queried through the *currentPrice* function (decreasing with time).

After the bidding is successful, the number of *token0* tokens exchanged will be recorded for the user. The final auction price of all users is the lowest price among participating users.

```
178  function bid(
179      // pool index
180      uint index,
181      // amount of token0 want to bid
182      uint amount0,
183      // amount of token1
184      uint amount1
185  ) external payable
186      nonReentrant
187      isPoolExist(index)
188      checkBotHolder(index)
189      isPoolNotClosed(index)
190  {
191      address payable sender = msg.sender;
192      require(tx.origin == msg.sender, "disallow contract caller");
193      if (enableWhiteList) {
194          require(whitelistP[index][sender], "sender not in whitelist");
195      }
196      Pool memory pool = pools[index];
197      require(pool.openAt <= now, "pool not open");
198      require(amount0 != 0, "the value of amount0 is zero");
199      require(amount1 != 0, "the value of amount1 is zero");
200      require(pool.amountTotal0 > amountSwap0P[index], "swap amount is zero");
201
202      // calculate price
203      uint curPrice = currentPrice(index);
204      uint bidPrice = amount1.mul(1 ether).div(amount0);
205      require(bidPrice >= curPrice, "the bid price is lower than the current price");
206
207      if (lowestBidPrice[index] == 0 || lowestBidPrice[index] > bidPrice) {
208          lowestBidPrice[index] = bidPrice;
209      }
210
211      address token1 = pool.token1;
212      if (token1 == address(0)) {
213          require(amount1 == msg.value, "invalid ETH amount");
214      } else {
215          IERC20(token1).safeTransferFrom(sender, address(this), amount1);
216      }
217
218      _swap(sender, index, amount0, amount1);
219
220      emit Bid(index, sender, amount0, amount1);
221  }
```

Figure 21 Source Code of *bid* Function

- Related Functions: *isPoolExist, checkBotHolder, currentPrice, isPoolNotClosed, safeTransferFrom, bid*
- Result: Pass

(4) Claim tokens

- Description: As shown in the figure below, the contract implements *creatorClaim* and *bidderClaim* functions for creators and users of the trading pool to withdraw tokens. The final auction price is the historical lowest price in the trading pool. If the tokens paid by the user exceed the historical lowest price, the excess will be returned to the user. If the user is paying in BNB, a 1.5% handling fee will be charged, and the handling fee will be transferred by calling the *depositReward* function in the *StakeContract* contract.

```
223     function creatorClaim(uint index) external
224         nonReentrant
225         isPoolExist(index)
226         isPoolClosed(index)
227     {
228         address payable creator = msg.sender;
229         require(isCreator(creator, index), "sender is not pool creator");
230         require(!creatorClaimedP[index], "creator has claimed this pool");
231         creatorClaimedP[index] = true;
232
233         // remove ownership of this pool from creator
234         delete myCreatedP[creator];
235
236         // calculate un-filled amount0
237         Pool memory pool = pools[index];
238         uint unFilledAmount0 = pool.amountTotal0.sub(amountSwap0P[index]);
239         if (unFilledAmount0 > 0) {
240             // transfer un-filled amount of token0 back to creator
241             IERC20(pool.token0).safeTransfer(creator, unFilledAmount0);
242         }
243
244         // send token1 to creator
245         uint amount1 = lowestBidPrice[index].mul(amountSwap0P[index]).div(1 ether);
246         if (amount1 > 0) {
247             if (pool.token1 == address(0)) {
248                 uint256 txFee = amount1.mul(getTxFeeRatio()).div(1 ether);
249                 uint256 _actualAmount1 = amount1.sub(txFee);
250                 if (_actualAmount1 > 0) {
251                     pool.creator.transfer(_actualAmount1);
252                 }
253                 if (txFee > 0) {
254                     // deposit transaction fee to staking contract
255                     IBounceStake(getStakeContract()).depositReward{value: txFee}();
256                 }
257             } else {
258                 IERC20(pool.token1).safeTransfer(pool.creator, amount1);
259             }
260         }
261
262         emit Claimed(index, creator, unFilledAmount0);
263     }
```

Figure 22 Source Code of *creatorClaim* Function

```
265  function bidderClaim(uint index) external
266      nonReentrant
267      isPoolExist(index)
268      isPoolClosed(index)
269  {
270      address payable bidder = msg.sender;
271      require(!bidderClaimedP[bidder][index], "bidder has claimed this pool");
272      bidderClaimedP[bidder][index] = true;
273
274      Pool memory pool = pools[index];
275      // send token0 to bidder
276      if (myAmountSwap0P[bidder][index] > 0) {
277          IERC20(pool.token0).safeTransfer(bidder, myAmountSwap0P[bidder][index]);
278      }
279
280      // send unfilled token1 to bidder
281      uint actualAmount1 = lowestBidPrice[index].mul(myAmountSwap0P[bidder][index]).div(1 ether);
282      uint unfilledAmount1 = myAmountSwap1P[bidder][index].sub(actualAmount1);
283      if (unfilledAmount1 > 0) {
284          if (pool.token1 == address(0)) {
285              bidder.transfer(unfilledAmount1);
286          } else {
287              IERC20(pool.token1).safeTransfer(bidder, unfilledAmount1);
288          }
289      }
290  }
```

Figure 23 Source Code of *bidderClaim* Function

- Related Functions: *creatorClaim, bidderClaim, isPoolExist, isPoolNotSwap, isCreator*
- Result: Pass

(5) Query functions

- Description: The contract implements functions such as *getPoolCount, currentPrice, getTxFeeRatio, getMinValueOfBotHolder, getBotToken and nextRoundInSeconds* to query the relevant data in the contract.
- Related Functions: *getPoolCount, currentPrice, getTxFeeRatio, getMinValueOfBotHolder, getBotToken, nextRoundInSeconds*
- Result: Pass

3.4 Business analysis of contract BounceFixedSwap

(1) Contract initialization

- Description: As shown in the figure below, the contract implements the *initialize* and *initialize_rinkeby* functions to initialize the contract (initialize only once). After the contract is deployed, any user can call these functions to set the value of the variable *BotToken*, the variable *MinValueOfBotHolder*, the variable *TxFeeRatio*, the variable *StakeContract,* etc.

```
102  function initialize() public initializer {
103      super.__Ownable_init();
104      super.__ReentrancyGuard_init();
105
106      config[TxFeeRatio] = 0.015 ether;
107      config[MinValueOfBotHolder] = 60 ether;
108
109      config[BotToken] = uint(0xA9B1Eb5908CfC3cdf91F9B8B3a74108598009096);
110      config[StakeContract] = uint(0x98945BC69A554F8b129b09aC8AfDc2cc2431c48E);
111  }
112
113  function initialize_rinkeby() public {
114      initialize();
115
116      config[BotToken] = uint(0x5E26FA0FE067d28aae8aFf2fB85Ac2E693BD9EfA);
117      config[StakeContract] = uint(0xa77A9FcbA2Ae5599e0054369d1655D186020ECE1);
118  }
```

Figure 24 Source Code of *initialize* and *initialize_rinkeby* Functions

- Related Functions: *initialize, initialize_rinkeby*
- Result: Pass

(2) Create a swap pool

- Description: As shown in the figure below, the contract implements the *create* function for users to create a swap pool. The creator can put any BEP-20 token as a commodity in the swap pool. When creating a swap pool, creator can choose whether to limit purchases to users who hold a certain amount of Bot tokens and set the auction start price, maximum price and other parameters. The creator can also decide who can make a purchase by setting a whitelist. The creator can set a claim delay, and the user can only claim the exchanged tokens after the time is up. The creator can also set the maximum amount that a single address can exchange.

```
120  function create(CreateReq memory poolReq, address[] memory whitelist_) external nonReentrant {
121      uint index = pools.length;
122      require(tx.origin == msg.sender, "disallow contract caller");
123      require(poolReq.amountTotal0 != 0, "invalid amountTotal0");
124      require(poolReq.amountTotal1 != 0, "invalid amountTotal1");
125      require(poolReq.openAt >= now, "invalid openAt");
126      require(poolReq.closeAt > poolReq.openAt, "invalid closeAt");
127      require(poolReq.claimAt == 0 || poolReq.claimAt >= poolReq.closeAt, "invalid closeAt");
128      require(bytes(poolReq.name).length <= 15, "length of name is too long");
129
130      if (poolReq.maxAmount1PerWallet != 0) {
131          maxAmount1PerWalletP[index] = poolReq.maxAmount1PerWallet;
132      }
```

Figure 25 Source Code of *create* Function (1/2)

```
133         if (poolReq.onlyBot) {
134             onlyBotHolderP[index] = poolReq.onlyBot;
135         }
136
137         // transfer amount of token0 to this contract
138         IERC20  _token0 = IERC20(poolReq.token0);
139         uint token0BalanceBefore = _token0.balanceOf(address(this));
140         _token0.safeTransferFrom(msg.sender, address(this), poolReq.amountTotal0);
141         require(
142             _token0.balanceOf(address(this)).sub(token0BalanceBefore) == poolReq.amountTotal0,
143             "not support deflationary token"
144         );
145
146         if (poolReq.enableWhiteList) {
147             require(whitelist_.length > 0, "no whitelist imported");
148             _addWhitelist(index, whitelist_);
149         }
150
151         Pool memory pool;
152         pool.name = poolReq.name;
153         pool.creator = msg.sender;
154         pool.token0 = poolReq.token0;
155         pool.token1 = poolReq.token1;
156         pool.amountTotal0 = poolReq.amountTotal0;
157         pool.amountTotal1 = poolReq.amountTotal1;
158         pool.openAt = poolReq.openAt;
159         pool.closeAt = poolReq.closeAt;
160         pool.claimAt = poolReq.claimAt;
161         pool.enableWhiteList = poolReq.enableWhiteList;
162         pools.push(pool);
163
164         emit Created(index, msg.sender, pool);
165     }
```

Figure 26 Source Code of *create* Function (2/2)

- Related Functions: *create, isPoolNotCreate, balanceOf, safeTransferFrom*
- Result: Pass

(3) Swap for BEP-20 tokens

- Description: As shown in the figure below, the contract implements the *swap* function for users to swap BEP-20 tokens in the designated swap pool. If the creator of the redemption pool sets a whitelist, only users in the whitelist can participate in the redemption. If the creator of the redemption pool sets a claim delay, the convertor can only claim it by calling the *userClaim* function after the claim time arrives. If it is not set, the account will be received in real time. If the token paid by the user is BNB, a 1.5%

handling fee will be charged. The handling fee is temporarily stored in the contract and processed when the *creatorClaim* function is called.

```
167     function swap(uint index, uint amount1) external payable
168         nonReentrant
169         isPoolExist(index)
170         isPoolNotClosed(index)
171         checkBotHolder(index)
172     {
173         address payable sender = msg.sender;
174         require(tx.origin == msg.sender, "disallow contract caller");
175         Pool memory pool = pools[index];
176
177         if (pool.enableWhiteList) {
178             require(whitelistP[index][sender], "sender not in whitelist");
179         }
180         require(pool.openAt <= now, "pool not open");
181         require(pool.amountTotal1 > amountSwap1P[index], "swap amount is zero");
182
183         // check if amount1 is exceeded
184         uint excessAmount1 = 0;
185         uint _amount1 = pool.amountTotal1.sub(amountSwap1P[index]);
186         if (_amount1 < amount1) {
187             excessAmount1 = amount1.sub(_amount1);
188         } else {
189             _amount1 = amount1;
190         }
191
192         // check if amount0 is exceeded
193         uint amount0 = _amount1.mul(pool.amountTotal0).div(pool.amountTotal1);
194         uint _amount0 = pool.amountTotal0.sub(amountSwap0P[index]);
195         if (_amount0 < amount0) {
196             require(amount0 - _amount0 > 100, "amount0 is too big");
197         } else {
198             _amount0 = amount0;
199         }
200
201         amountSwap0P[index] = amountSwap0P[index].add(_amount0);
202         amountSwap1P[index] = amountSwap1P[index].add(_amount1);
203         myAmountSwapped0[sender][index] = myAmountSwapped0[sender][index].add(_amount0);
204         // check if swapped amount of token1 is exceeded maximum allowance
205         if (maxAmount1PerWalletP[index] != 0) {
206             require(
207                 myAmountSwapped1[sender][index].add(_amount1) <= maxAmount1PerWalletP[index],
208                 "swapped amount of token1 is exceeded maximum allowance"
209             );
210             myAmountSwapped1[sender][index] = myAmountSwapped1[sender][index].add(_amount1);
211         }
```

Figure 27 Source Code of *swap* Function (1/2)

```
213         if (pool.amountTotal1 == amountSwap1P[index]) {
214             filledAtP[index] = now;
215         }
216
217         // transfer amount of token1 to this contract
218         if (pool.token1 == address(0)) {
219             require(msg.value == amount1, "invalid amount of ETH");
220         } else {
221             IERC20(pool.token1).safeTransferFrom(sender, address(this), amount1);
222         }
223
224         if (pool.claimAt == 0) {
225             if (_amount0 > 0) {
226                 // send token0 to sender
227                 if (pool.token0 == address(0)) {
228                     sender.transfer(_amount0);
229                 } else {
230                     IERC20(pool.token0).safeTransfer(sender, _amount0);
231                 }
232             }
233         }
234         if (excessAmount1 > 0) {
235             // send excess amount of token1 back to sender
236             if (pool.token1 == address(0)) {
237                 sender.transfer(excessAmount1);
238             } else {
239                 IERC20(pool.token1).safeTransfer(sender, excessAmount1);
240             }
241         }
242
243         // send token1 to creator
244         uint256 txFee = _amount1.mul(getTxFeeRatio()).div(1 ether);
245         txFeeP[index] = txFeeP[index].add(txFee);
246         uint256 _actualAmount1 = _amount1.sub(txFee);
247         if (_actualAmount1 > 0) {
248             if (pool.token1 == address(0)) {
249                 pool.creator.transfer(_actualAmount1);
250             } else {
251                 IERC20(pool.token1).safeTransfer(pool.creator, _actualAmount1);
252             }
253         }
254
255         emit Swapped(index, sender, _amount0, _actualAmount1, txFee);
256     }
```

Figure 28 Source Code of *swap* Function (2/2)

- Related Functions: swap, *isPoolExist, checkBotHolder, safeTransferFrom, isPoolNotClosed, getTxFeeRatio, safeTransfer*
- Safety Recommendation: When the user calls the *bid* function to make a bid, no matter what token the user pays, a handling fee is charged. When the creator receives it, the handling fee is only processed for the case of BNB as *token1*. It is recommended to change the handling fee collection code, and only charge the handling fee when *token1* is a zero address.
- Fixed Result: Fixed.

```
167  function swap(uint index, uint amount1) external payable
168      nonReentrant
169      isPoolExist(index)
170      isPoolNotClosed(index)
171      checkBotHolder(index)
172  {
173      address payable sender = msg.sender;
174      require(tx.origin == msg.sender, "disallow contract caller");
175      Pool memory pool = pools[index];
176
177      if (pool.enableWhiteList) {
178          require(whitelistP[index][sender], "sender not in whitelist");
179      }
180      require(pool.openAt <= now, "pool not open");
181      require(pool.amountTotal1 > amountSwap1P[index], "swap amount is zero");
182
183      // check if amount1 is exceeded
184      uint excessAmount1 = 0;
185      uint _amount1 = pool.amountTotal1.sub(amountSwap1P[index]);
186      if (_amount1 < amount1) {
187          excessAmount1 = amount1.sub(_amount1);
188      } else {
189          _amount1 = amount1;
190      }
191
192      // check if amount0 is exceeded
193      uint amount0 = _amount1.mul(pool.amountTotal0).div(pool.amountTotal1);
194      uint _amount0 = pool.amountTotal0.sub(amountSwap0P[index]);
195      if (_amount0 > amount0) {
196          _amount0 = amount0;
197      }
198
199      amountSwap0P[index] = amountSwap0P[index].add(_amount0);
200      amountSwap1P[index] = amountSwap1P[index].add(_amount1);
```

Figure 29 Source Code of *swap* Function (1/2)

```
201     myAmountSwapped0[sender][index] = myAmountSwapped0[sender][index].add(_amount0);
202     // check if swapped amount of token1 is exceeded maximum allowance
203     if (maxAmount1PerWalletP[index] != 0) {
204       require(
205         myAmountSwapped1[sender][index].add(_amount1) <= maxAmount1PerWalletP[index],
206         "swapped amount of token1 is exceeded maximum allowance"
207       );
208       myAmountSwapped1[sender][index] = myAmountSwapped1[sender][index].add(_amount1);
209     }
210
211     if (pool.amountTotal1 == amountSwap1P[index]) {
212       filledAtP[index] = now;
213     }
214
215     // transfer amount of token1 to this contract
216     if (pool.token1 == address(0)) {
217       require(msg.value == amount1, "invalid amount of ETH");
218     } else {
219       IERC20(pool.token1).safeTransferFrom(sender, address(this), amount1);
220     }
221
222     if (pool.claimAt == 0) {
223       if (_amount0 > 0) {
224         // send token0 to sender
225         IERC20(pool.token0).safeTransfer(sender, _amount0);
226       }
227     }
228     if (excessAmount1 > 0) {
229       // send excess amount of token1 back to sender
230       if (pool.token1 == address(0)) {
231         sender.transfer(excessAmount1);
232       } else {
233         IERC20(pool.token1).safeTransfer(sender, excessAmount1);
234       }
235     }
236
237     // send token1 to creator
238     uint256 txFee = 0;
239     uint256 _actualAmount1 = _amount1;
240     if (pool.token1 == address(0)) {
241       txFee = _amount1.mul(getTxFeeRatio()).div(1 ether);
242       txFeeP[index] = txFeeP[index].add(txFee);
243       _actualAmount1 = _amount1.sub(txFee);
244       pool.creator.transfer(_actualAmount1);
245     } else {
246       IERC20(pool.token1).safeTransfer(pool.creator, _actualAmount1);
247     }
248
249     emit Swapped(index, sender, _amount0, _actualAmount1, txFee);
250   }
```

Figure 30 Source Code of swap Function (2/2)

- Result: Pass

(4) Claim tokens

- Description: As shown in the figure below, the contract implements *creatorClaim* and *userClaim* functions for creators and users of the swap pool to withdraw tokens. When the creator calls the *creatorClaim* function(In fact, any user can call this function), he transfers all the handling fee BNB charged by the pool to the contract by calling the *depositReward* function of the *StakeContract* contract, and receives the unpurchased *token0* in the pool. If the creator sets a claim delay, the user can claim the purchased tokens after the claim time is reached by calling the *userClaim* function.

```
258     function creatorClaim(uint index) external
259         nonReentrant
260         isPoolExist(index)
261         isPoolClosed(index)
262     {
263         Pool memory pool = pools[index];
264         require(!creatorClaimed[pool.creator][index], "claimed");
265         creatorClaimed[pool.creator][index] = true;
266 
267         if (txFeeP[index] > 0) {
268             if (pool.token1 == address(0)) {
269                 // deposit transaction fee to staking contract
270                 IBounceStake(getStakeContract()).depositReward{value: txFeeP[index]}();
271             }
272         }
273 
274         uint unSwapAmount0 = pool.amountTotal0 - amountSwap0P[index];
275         if (unSwapAmount0 > 0) {
276             IERC20(pool.token0).safeTransfer(pool.creator, unSwapAmount0);
277         }
278 
279         emit Claimed(index, msg.sender, unSwapAmount0, txFeeP[index]);
280     }
```

Figure 31 Source Code of *creatorClaim* Function

```
282     function userClaim(uint index) external
283         nonReentrant
284         isPoolExist(index)
285         isClaimReady(index)
286     {
287         Pool memory pool = pools[index];
288         address sender = msg.sender;
289         require(!myClaimed[sender][index], "claimed");
290         myClaimed[sender][index] = true;
291         if (myAmountSwapped0[sender][index] > 0) {
292             // send token0 to sender
293             if (pool.token0 == address(0)) {
294                 msg.sender.transfer(myAmountSwapped0[sender][index]);
295             } else {
296                 IERC20(pool.token0).safeTransfer(msg.sender, myAmountSwapped0[sender][index]);
297             }
298         }
299         emit UserClaimed(index, sender, myAmountSwapped0[sender][index]);
300     }
```

Figure 32 Source Code of *userClaim* Function

- Related Functions: *creatorClaim, userClaim, isPoolExist, isClaimReady, isCreator, isPoolClosed*
- Safety Recommendation: When the pool is created, *token0* cannot be a zero address, so the judgment of *token0* is a redundant code. It is recommended to delete.
- Fixed Result: Fixed.

```
276     function userClaim(uint index) external
277         nonReentrant
278         isPoolExist(index)
279         isClaimReady(index)
280     {
281         Pool memory pool = pools[index];
282         address sender = msg.sender;
283         require(!myClaimed[sender][index], "claimed");
284         myClaimed[sender][index] = true;
285         if (myAmountSwapped0[sender][index] > 0) {
286             // send token0 to sender
287             IERC20(pool.token0).safeTransfer(msg.sender, myAmountSwapped0[sender][index]);
288         }
289         emit UserClaimed(index, sender, myAmountSwapped0[sender][index]);
290     }
```

Figure 33 Source Code of *userClaim* Function(Fixed)

- Result: Pass

(5) Query functions

- Description: The contract implements functions such as *getPoolCount, getTxFeeRatio, getMinValueOfBotHolder, getBotToken* and *getStakeContract*to query the relevant data in the contract.
- Related Functions: *getPoolCount, getTxFeeRatio, getMinValueOfBotHolder, getBotToken, getStakeContract*
- Result: Pass

3.5 Business analysis of contract BounceLottery

(1) Contract initialization

- Description: As shown in the figure below, the contract implements the *initialize* function to initialize the contract (initialize only once). After the contract is deployed, any user can call this function to initialize the owner.

```
94    function initialize() public initializer {
95        super.__Ownable_init();
96    }
```

Figure 34 Source Code of *initialize* Function

- Related Functions: *initialize*
- Result: Pass

(2) Create a swap pool

- Description: As shown in the figure below, the contract implements the *create* function for users to create a swap pool. The creator can put any BEP-20 token as a commodity in the swap pool. When creating a swap pool, the creator can also decide who can make a purchase by setting a whitelist. The contract implements the *creatorClaim* and *playerClaim* functions for the creator and user of the pool to receive specified tokens. The creator can also set parameters such as the number of winners and the maximum number of participants, etc.

```
98  function create(CreateReq memory poolReq, address[] memory whitelist_) external
99      nonReentrant
100     nameNotBeenToken(poolReq.name)
101 {
102     require(tx.origin == msg.sender, "disallow contract caller");
103     require(poolReq.amountTotal0 >= poolReq.nShare, "amountTotal0 less than nShare");
104     require(poolReq.amountTotal1 != 0, "the value of amountTotal1 is zero");
105     require(poolReq.nShare != 0, "the value of nShare is zero");
106     require(poolReq.nShare <= poolReq.maxPlayer, "max player less than nShare");
107     require(poolReq.maxPlayer < 65536, "max player must less 65536");
108     require(poolReq.maxPlayer > 0, "the value of maxPlayer is zero");
109     require(poolReq.openAt <= poolReq.closeAt && poolReq.closeAt.sub(poolReq.openAt) < 7 days, "invalid closed");
110     require(bytes(poolReq.name).length <= 15, "the length of name is too long");
111
112     uint index = pools.length;
113
114     // transfer amount of token0 to this contract
115     IERC20  _token0 = IERC20(poolReq.token0);
116     uint token0BalanceBefore = _token0.balanceOf(address(this));
117     _token0.safeTransferFrom(poolReq.creator, address(this), poolReq.amountTotal0);
118     require(
119         _token0.balanceOf(address(this)).sub(token0BalanceBefore) == poolReq.amountTotal0,
120         "not support deflationary token"
121     );
122
123     if (poolReq.enableWhiteList) {
124         require(whitelist_.length > 0, "no whitelist imported");
125         _addWhitelist(index, whitelist_);
126     }
127
128     // creator pool
129     Pool memory pool;
130     pool.creator = poolReq.creator;
131     pool.name = poolReq.name;
132     pool.token0 = poolReq.token0;
133     pool.amountTotal0 = poolReq.amountTotal0;
134     pool.token1 = poolReq.token1;
135     pool.amountTotal1 = poolReq.amountTotal1;
136     pool.maxPlayer = poolReq.maxPlayer;
137     pool.openAt = poolReq.openAt;
138     pool.closeAt = poolReq.closeAt;
139     pool.enableWhiteList = poolReq.enableWhiteList;
```

Figure 35 Source Code of *create* Function (1/2)

```
142         pool.poolId = index;
143         pool.curPlayer = 0;
144         pool.claim = false;
145         pools.push(pool);
146         myCreate[poolReq.creator].push(index);
147         name2Id[poolReq.name] = index;
148         poolsExt[index].nShare = poolReq.nShare;
149
150         emit Created(pool);
151     }
```

Figure 36 Source Code of *create* Function (2/2)

- Related Functions: *create, nameNotBeenToken, isContract, safeTransferFrom, balanceOf*
- Result: Pass

(3) Bet for BEP-20 tokens

- Description: As shown in the figure below, the contract implements the *bet* function for users to participate in the lottery. Each user can only participate once. Each user will update the corresponding information in the pool.

```
625     function bet(
626         uint index
627     ) external payable
628     nonReentrant
629     isPoolExist(index)
630     isPoolNotClosed(index)
631     {
632         address payable sender = msg.sender;
633         uint ethAmount1 = msg.value;
634         Pool memory pool = pools[index];
635         require(allPlayer[index][sender] == 0, "You have already bet");
636
637         if (pool.enableWhiteList) {
638             require(whitelistP[index][sender], "sender not in whitelist");
639         }
640
641         //require(pool.creator != sender, "creator can't bid the pool created by self");
642
643         require(pool.curPlayer < pool.maxPlayer, "Player has reached the upper limit");
644         if (pool.token1 == address(0)) {
645             require(ethAmount1 >= pool.amountTotal1, "The bet amount is too low");
646         } else {
647             IERC20(pool.token1).safeTransferFrom(sender, address(this), pool.amountTotal1);
648         }
```

Figure 37 Source Code of *bet* Function (1/2)

```
649     allPlayer[index][sender] = pools[index].curPlayer + 1;
650     pools[index].curPlayer += 1;
651
652     myPlay[sender].push(index);
653
654     poolsExt[index].lastHash = uint(keccak256(abi.encodePacked(block.timestamp, block.difficulty, blockhash
        (block.number - 1))));
655
656     emit Bet(sender, index);
657 }
```

Figure 38 Source Code of *bet* Function (2/2)

- Related Functions: bet, *isContract, nonReentrant, isPoolExist, isPoolNotClosed, safeTransferFrom*
- Safety Recommendation: The parameters used to calculate the winner are all directly obtainable. After the purchase, user can calculate whether user are currently winning. If user use the contract to operate, user can make a purchases in one transaction. If the purchase does not win, user can call revert roll back this transaction. It is recommended to prohibit the contract as a buyer.
- Fixed Result: Fixed.

```
185 function bet(
186     uint index
187 ) external payable
188     nonReentrant
189     isPoolExist(index)
190     isPoolNotClosed(index)
191 {
192     address sender = msg.sender;
193     require(tx.origin == msg.sender, "disallow contract caller");
194     Pool memory pool = pools[index];
195     require(allPlayer[index][sender] == 0, "You have already bet");
196
197     if (pool.enableWhiteList) {
198         require(whitelistP[index][sender], "sender not in whitelist");
199     }
200
201     require(pool.curPlayer < pool.maxPlayer, "Player has reached the upper limit");
202     if (pool.token1 == address(0)) {
203         require(msg.value == pool.amountTotal1, "The bet amount is too low");
204     } else {
205         IERC20(pool.token1).safeTransferFrom(sender, address(this), pool.amountTotal1);
206     }
207     allPlayer[index][sender] = pools[index].curPlayer + 1;
208     pools[index].curPlayer += 1;
```

Figure 39 Source Code of *bet* Function (1/2)

```
210         myPlay[sender].push(index);
211
212         poolsExt[index].lastHash = uint(keccak256(abi.encodePacked(block.timestamp, block.difficulty, blockhash
            (block.number - 1))));
213
214         emit Bet(sender, index);
215     }
```

Figure 40 Source Code of *bet* Function (2/2)

- Result: Pass

(4) Claim tokens

- Description: As shown in the figure below, the contract implements the *creatorClaim* and *playerClaim* functions for the creator and user of the pool to receive specified tokens. If the number of winners does not reach the maximum when the creator receives it, part of the unpurchased tokens will be returned to the creator. When the user receives it, he can judge whether he has won the prize by calling the *isWinner* function. If he does not win the prize, the paid tokens will be returned to the user. If the prize is won, the purchased tokens will be transferred to the user.

```
217 function creatorClaim(uint index) external
218     nonReentrant
219     isPoolExist(index)
220     isPoolClosed(index)
221 {
222     address payable sender = msg.sender;
223     require(isCreator(sender, index), "sender is not pool creator");
224
225     Pool memory pool = pools[index];
226     require(!pool.claim, "creator has claimed this pool");
227
228     pools[index].claim = true;
229     if (pool.curPlayer == 0) {
230         IERC20(pool.token0).safeTransfer(sender, pool.amountTotal0);
231         emit Claimed(sender, index);
232         return;
233     }
234     uint nShare = poolsExt[index].nShare;
235     uint hitShare = (pool.curPlayer > nShare ? nShare : pool.curPlayer);
236     if (pool.token1 == address(0)) {
237         sender.transfer(pool.amountTotal1.mul(hitShare));
238     } else {
239         IERC20(pool.token1).safeTransfer(sender, pool.amountTotal1.mul(hitShare));
240     }
241     if (nShare > pool.curPlayer) {
242         IERC20(pool.token0).safeTransfer(sender, pool.amountTotal0.div(nShare).mul(nShare.sub(pool.curPlayer))
            );
243     }
244
245     emit Claimed(sender, index);
246 }
```

Figure 41 Source Code of *creatorClaim* Function

```
248  function playerClaim(uint index) external
249      nonReentrant
250      isPoolExist(index)
251      isPoolClosed(index)
252  {
253      address payable sender = msg.sender;
254      require(allPlayer[index][sender] > 0, "You haven't bet yet");
255      require(!isCreator(sender, index), "sender is pool creator");
256
257      Pool memory pool = pools[index];
258
259      require(!allPlayerClaim[index][sender], "You have claimed this pool");
260      require(pool.closeAt < now, "It's not time to start the prize");
261      allPlayerClaim[index][sender] = true;
262
263      if (isWinner(index, address(sender))) {
264          IERC20(pool.token0).safeTransfer(sender, pool.amountTotal0.div(poolsExt[index].nShare));
265      } else {
266          if (pool.token1 == address(0)) {
267              sender.transfer(pool.amountTotal1);
268          } else {
269              IERC20(pool.token1).safeTransfer(sender, pool.amountTotal1);
270          }
271      }
272      emit Claimed(sender, index);
273  }
```

Figure 42 Source Code of *playerClaim* Function

- Related Functions: *creatorClaim, nonReentrant, isPoolExist, isPoolClosed, isCreator, transfer, safeTransfer, isWinner, calcRet, lo2*
- Result: Pass

(5) Query functions

- Description: The contract implements functions such as *getLotteryPoolInfo, getPlayerStatus, getPoolCount, isCreator* and *isWinner* to query the relevant data in the contract.
- Related Functions: *getLotteryPoolInfo, getPlayerStatus, getPoolCount, isCreator, isWinner*
- Result: Pass

3.6 Business analysis of contract BounceSealedBid

(1) Contract initialization

- Description: As shown in the figure below, the contract implements the *initialize* and *initialize_rinkeby* functions to initialize the contract (initialize only once). After the contract is deployed, any user can call these functions to set the value of the variable *BotToken*, the variable *MinValueOfBotHolder*, the variable *TxFeeRatio*, the variable *StakeContract,* etc.

```
107  function initialize() public initializer {
108      super.__Ownable_init();
109      super.__ReentrancyGuard_init();
110
111      config[TxFeeRatio] = 0.015 ether;
112      config[MinValueOfBotHolder] = 60 ether;
113      config[MaxBidCount] = 1000;
114
115      // mainnet
116      config[BotToken] = uint(0xA9B1Eb5908CfC3cdf91F9B8B3a74108598009096);
117      config[StakeContract] = uint(0x98945BC69A554F8b129b09aC8AfDc2cc2431c48E);
118  }
119
120  function initialize_rinkeby() public {
121      initialize();
122
123      config[BotToken] = uint(0x5E26FA0FE067d28aae8aFf2fB85Ac2E693BD9EfA);
124      config[StakeContract] = uint(0xa77A9FcbA2Ae5599e0054369d1655D186020ECE1);
125  }
```

Figure 43 Source Code of *initialize* and *initialize_rinkeby* Functions

- Related Functions: *initialize, initialize_rinkeby*
- Result: Pass

(2) Create a swap pool

- Description: As shown in the figure below, the contract implements the *create* function for users to create a swap pool. The creator can put any BEP-20 token as a commodity in the swap pool. When creating a swap pool, creator can choose whether to limit purchases to users who hold a certain amount of Bot tokens and set the auction start price, maximum price and other parameters. The creator can also decide who can make a purchase by setting a whitelist.

```
127  function create(CreateReq memory poolReq, address[] memory whitelist_) external nonReentrant {
128      require(tx.origin == msg.sender, "disallow contract caller");
129      require(poolReq.amountTotal0 != 0, "the value of amountTotal0 is zero");
130      require(poolReq.amountMin1 != 0, "the value of amountMin1 is zero");
131      require(poolReq.openAt <= poolReq.closeAt && poolReq.closeAt.sub(poolReq.openAt) < 7 days, "invalid
         closed");
132      require(bytes(poolReq.name).length <= 15, "length of name is too long");
133
134      uint index = pools.length;
135
136      // transfer amount of token0 to this contract
137      IERC20  _token0 = IERC20(poolReq.token0);
138      uint token0BalanceBefore = _token0.balanceOf(address(this));
139      _token0.safeTransferFrom(poolReq.creator, address(this), poolReq.amountTotal0);
```

Figure 44 Source Code of *create* Function (1/2)

```
140         require(
141             _token0.balanceOf(address(this)).sub(token0BalanceBefore) == poolReq.amountTotal0,
142             "not support deflationary token"
143         );
144
145         if (poolReq.enableWhiteList) {
146             require(whitelist_.length > 0, "no whitelist imported");
147             _addWhitelist(index, whitelist_);
148         }
149
150         // creator pool
151         Pool memory pool;
152         pool.name = poolReq.name;
153         pool.creator = poolReq.creator;
154         pool.token0 = poolReq.token0;
155         pool.token1 = poolReq.token1;
156         pool.amountTotal0 = poolReq.amountTotal0;
157         pool.amountMin1 = poolReq.amountMin1;
158         pool.openAt = poolReq.openAt;
159         pool.closeAt = poolReq.closeAt;
160         pool.enableWhiteList = poolReq.enableWhiteList;
161         pools.push(pool);
162
163         if (poolReq.maxAmount1PerWallet != 0) {
164             maxAmount1PerWallet[index] = poolReq.maxAmount1PerWallet;
165         }
166         if (poolReq.onlyBot) {
167             onlyBotHolderP[index] = poolReq.onlyBot;
168         }
169
170         myCreatedP[poolReq.creator] = pools.length;
171 //          bidderListHeaderP[index] = type(uint).max;
172
173         emit Created(index, msg.sender, pool);
174     }
```

Figure 45 Source Code of *create* Function (2/2)

- Related Functions: *create, isPoolNotCreate, balanceOf, safeTransferFrom*
- Result: Pass

(3) Bid for BEP-20 tokens

- Description: As shown in the figure below, the contract implements the *bid* function for users to participate in the auction. By calling the *bid* function, the user enters the pool that needs to participate in the auction and the number of *token1* to be paid and the number of *token0* to be purchased (must be

greater than the minimum value set by the creator). The user bid cannot be less than the minimum bid. Each time a user calls the *bid* function, the corresponding data is updated, and the bids of all users are sorted from largest to smallest.

```
176  function bid(
177      // pool index
178      uint index,
179      // amount of token0 want to bid
180      uint amount0,
181      // amount of token1
182      uint amount1
183  ) external payable
184      nonReentrant
185      isPoolExist(index)
186      checkBotHolder(index)
187      isPoolNotClosed(index)
188  {
189      address sender = msg.sender;
190      require(tx.origin == msg.sender, "disallow contract caller");
191      Pool memory pool = pools[index];
192      if (pool.enableWhiteList) {
193          require(whitelistP[index][sender], "sender not in whitelist");
194      }
195      require(pool.openAt <= now, "pool not open");
196      require(amount0 != 0, "the value of amount0 is zero");
197      require(amount1 != 0, "the value of amount1 is zero");
198      require(myAmountBid1P[sender][index] == 0, "this pool has been bid by this sender");
199      require(amount0.mul(getMaxBidCount()) >= pool.amountTotal0, "the bid amount is too low");
200      require(amount1 >= maxAmount1PerWallet[index], "the bid amount is lower than minimum ETH");
201
202      // calculate price
203      uint minPrice = pool.amountMin1.mul(1 ether).div(pool.amountTotal0);
204      uint price = amount1.mul(1 ether).div(amount0);
205      require(price >= minPrice, "your bid price is lower than the minimum price");
206
207      address token1 = pool.token1;
208      if (token1 == address(0)) {
209          require(amount1 == msg.value, "invalid ETH amount");
210      } else {
211          IERC20(token1).safeTransferFrom(sender, address(this), amount1);
212      }
213
214      // record pool index
215      myBidP[sender].push(index);
216      myPrice[sender][index] = price;
217      myAmountBid0P[sender][index] = amount0;
218      myAmountBid1P[sender][index] = amount1;
219      bidCountP[index]++;
220
221      // check if the sorted bidder list can fill the pool
222      adjustBidderList(sender, index, price);
223
224      emit Bid(index, sender, amount0, amount1);
225  }
```

Figure 46 Source Code of *swap* Function

- Related Functions: bid, *nonReentrant, isPoolExist, checkBotHolder, isPoolNotClosed, getMaxBidCount, safeTransferFrom*
- Result: Pass

(4) Claim tokens

- Description: As shown in the figure below, the contract implements *creatorClaim* and *bidderClaim* functions for users and creators to receive corresponding tokens. When the creator receives it, the *creatorFilledAmount* function is called to query the quantity that has been purchased. If not all are purchased, the remaining part will be returned to the creator, and if the *token0* set by the creator is BNB, a 1.5% handling fee will be charged The fee will be transferred to the contract by calling the *depositReward* function of the *StakeContract* contract. When the user calls *bidderClaim* function to claim, he will use *bidderFilledAmount* function to calculate the amount he has purchased. If the paid tokens are not all consumed, the remaining part will be returned to the user. When the creator and user calculate the purchase quantity, they are calculated according to the recorded price from high to low.

```
227  function creatorClaim(uint index) external
228      nonReentrant
229      isPoolExist(index)
230      isPoolClosed(index)
231  {
232      address payable creator = msg.sender;
233      Pool memory pool = pools[index];
234      require(pool.creator == creator, "sender is not pool creator");
235      require(!creatorClaimedP[index], "creator has claimed this pool");
236      creatorClaimedP[index] = true;
237
238      // remove ownership of this pool from creator
239      delete myCreatedP[creator];
240
241      (uint filledAmount0, uint filledAmount1) = creatorFilledAmount(index);
242      // calculate un-filled amount0
243      uint unFilledAmount0 = pool.amountTotal0.sub(filledAmount0);
244      if (unFilledAmount0 > 0) {
245          // transfer un-filled amount of token0 back to creator
246          IERC20(pool.token0).safeTransfer(creator, unFilledAmount0);
247      }
248
249      uint actualAmount1 = filledAmount1;
250      if (pool.token1 == address(0)) {
251          // calculate transaction fee;
252          uint txFee = filledAmount1.mul(getTxFeeRatio()).div(1 ether);
253          // calculate actual amount1;
254          actualAmount1 = filledAmount1.sub(txFee);
```

Figure 47 Source Code of *creatorClaim* Function (1/2)

```
255         if (actualAmount1 > 0) {
256             // transfer actual amount of token1 to creator
257             creator.transfer(actualAmount1);
258         }
259         if (txFee > 0) {
260             // deposit transaction fee to staking contract
261             IBounceStake(getStakeContract()).depositReward{value: txFee}();
262         }
263     } else {
264         IERC20(pool.token1).safeTransfer(creator, actualAmount1);
265     }
266
267     emit CreatorClaimed(index, creator, unFilledAmount0, actualAmount1);
268 }
```

Figure 48 Source Code of *creatorClaim* Function (2/2)

```
270 function bidderClaim(uint index) external
271     nonReentrant
272     isPoolExist(index)
273     isPoolClosed(index)
274 {
275     Pool memory pool = pools[index];
276     address payable sender = msg.sender;
277     require(!myClaimedP[sender][index], "sender has claimed this pool");
278     require(
279         myAmountBid1P[sender][index] > 0 && bidderListHeaderP[index] != type(uint).max,
280         "sender didn't bid this pool"
281     );
282     myClaimedP[sender][index] = true;
283
284     (uint filledAmount0, uint filledAmount1) = bidderFilledAmount(sender, index);
285     uint unFilledAmount1 = myAmountBid1P[sender][index].sub(filledAmount1);
286
287     if (filledAmount0 > 0) {
288         // transfer filled amount of token0 to bidder
289         IERC20(pool.token0).safeTransfer(sender, filledAmount0);
290     }
291     if (unFilledAmount1 > 0) {
292         // transfer un-filled amount of token1 back to bidder
293         if (pool.token1 == address(0)) {
294             sender.transfer(unFilledAmount1);
295         } else {
296             IERC20(pool.token1).safeTransfer(sender, unFilledAmount1);
297         }
298     }
299
300     emit BidClaimed(index, sender, filledAmount0, unFilledAmount1);
301 }
```

Figure 49 Source Code of *userClaim* Function

- Related Functions: *creatorClaim, creatorFilledAmount, safeTransfer, nonReentrant, isPoolExist, isPoolClosed, getTxFeeRatio, getStakeContract, depositReward, bidderClaim, bidderFilledAmount, transfer*
- Result: Pass

(5) Query functions

- Description: The contract implements functions such as *getMyBidPools, getBidderListCount, getMyBidCount, getTxFeeRatio, getMaxBidCountand getMinValueOfBotHolder, getBotToken, getStakeContract* and *getPoolCount* to query the relevant data in the contract.
- Related Functions: *getMyBidPools, getBidderListCount, getMyBidCount, getTxFeeRatio, getMaxBidCountand getMinValueOfBotHolder, getBotToken, getStakeContract, getPoolCount*
- Result: Pass

3.7 Other audit recommendations

- Description: When *token0* or *token1* are deflationary tokens, an exception will be thrown during settlement and cannot be traded; when *token0* or *token1* are inflation tokens, the excess tokens will remain in the contract and cannot be taken out. It is recommended to restrict the types of tokens through the token whitelist.
- Fixed Result: Ignored. The project party replied: The currency will be restricted at the front end, and the contract will not consider adding restrictions for the time being.

4. Conclusion

Beosin(Chengdu LianAn) conducted a detailed audit on the design and code implementation of the smart contracts project BounceNFT. The problems found by the audit team during the audit process have been notified to the project party and reached an agreement on the repair results, the overall audit result of the smart contracts project BounceNFT is **Pass**.

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com